END

AD-A182 063

②

AFWAL-TR-86-4006
Volume VII
Part 1

INTEGRATED INFORMATION
SUPPORT SYSTEM (IISS)
Volume VII - Communications Subsystem
Part 1 - COMM Development Specification

General Electric Company
Production Resources Consulting
One River Road
Schenectady, New York 12345

DTIC
ELECTE
JUL 0 6 1987
S
D

Final Report for Period 22 September 1980 - 31 July 1985

November 1985

Approved for public release; distribution is unlimited.

PREPARED FOR:

MATERIALS LABORATORY
AIR FORCE WRIGHT AERONAUTICAL LABORATORIES
AIR FORCE SYSTEMS COMMAND
WRIGHT-PATTERSON AFB, OH 45433-6533

87   7   2   038

NOTICE

This technical report has been reviewed and is approved for publication.

DAVID L. JUDSON, PROJECT MANAGER
AFWAL/MLTC
WRIGHT PATTERSON AFB OH 45433

DATE 5 Aug 1986

FOR THE COMMANDER:

GERALD C. SHUMAKER, BRANCH CHIEF
AFWAL/MLTC
WRIGHT PATTERSON AFB OH 45433

DATE 7 Aug 86

REPORT DOCUMENTATION PAGE

1a. REPORT SECURITY CLASSIFICATION		1b. RESTRICTIVE MARKINGS			
Unclassified					

2a. SECURITY CLASSIFICATION AUTHORITY		3. DISTRIBUTION/AVAILABILITY OF REPORT			
		Approved for public release; distribution is unlimited.			
2b. DECLASSIFICATION/DOWNGRADING SCHEDULE					

4. PERFORMING ORGANIZATION REPORT NUMBER(S)		5. MONITORING ORGANIZATION REPORT NUMBER(S)			
		AFWAL-TR-86-40006 Vol VII, Part 1			

6a. NAME OF PERFORMING ORGANIZATION	6b. OFFICE SYMBOL (If applicable)	7a. NAME OF MONITORING ORGANIZATION			
General Electric Company Production Resources Consulting		AFWAL/MLTC			

6c. ADDRESS (City, State and ZIP Code)		7b. ADDRESS (City, State and ZIP Code)			
1 River Road Schenectady, NY 12345		WPAFB, OH 45433-6533			

8a. NAME OF FUNDING/SPONSORING ORGANIZATION	8b. OFFICE SYMBOL (If applicable)	9. PROCUREMENT INSTRUMENT IDENTIFICATION NUMBER			
Materials Laboratory Air Force Systems Command, USAF	AFWAL/MLTC	F33615-80-C-5155			

8c. ADDRESS (City, State and ZIP Code)		10. SOURCE OF FUNDING NOS			
		PROGRAM ELEMENT NO.	PROJECT NO.	TASK NO.	WORK UNIT NO.
Wright-Patterson AFB, Ohio 45433		78011F	7500	62	01

11. TITLE (Include Security Classification)					
(See Reverse)					

12. PERSONAL AUTHOR(S) Weldon, Karen and Phillips, Donna

13a. TYPE OF REPORT	13b. TIME COVERED	14. DATE OF REPORT (Yr., Mo., Day)	15. PAGE COUNT
Final Technical Report	22 Sept 1980 - 31 July 1985	1985 November	64

16. SUPPLEMENTARY NOTATION	
ICAM Project Priority 6201	The computer software contained herein are theoretical and/or references that in no way reflect Air Force-owned or -developed computer software.

17. COSATI CODES			18. SUBJECT TERMS (Continue on reverse if necessary and identify by block number)
FIELD	GROUP	SUB. GR.	
1305	0905		

19. ABSTRACT (Continue on reverse if necessary and identify by block number)

This document defines the Development Specification for the Communications Subsystem (COMM) configuration item of the Integrated Information Support System (IISS) Test Bed. The Test Bed is an integrated hardware/software environment in which various Integrated Computer Aided Manufacturing (ICAM) Program applications will function cooperatively.

20. DISTRIBUTION/AVAILABILITY OF ABSTRACT	21. ABSTRACT SECURITY CLASSIFICATION		
UNCLASSIFIED/UNLIMITED ☒ SAME AS RPT. ☐ DTIC USERS ☐	Unclassified		

22a. NAME OF RESPONSIBLE INDIVIDUAL	22b. TELEPHONE NUMBER (Include Area Code)	22c. OFFICE SYMBOL
David L. Judson	513-255-6976	AFWAL/MLTC

DD FORM 1473, 83 APR   EDITION OF 1 JAN 73 IS OBSOLETE.

11. Title

Integrated Information Support System (IISS)
Vol VII - Communications Subsystem
Part 1 - COMM Development Specification

PREFACE

This development specification covers the work performed under Air Force Contract F33615-80-C-5155 (ICAM Project 6201). This contract is sponsored by the Materials Laboratory, Air Force Systems Command, Wright-Patterson Air Force Base, Ohio. It was administered under the technical direction of Mr. Gerald C. Shumaker, ICAM Program Manager, Manufacturing Technology Division, through Project Manager, Mr. David Judson. The Prime Contractor was Production Resources Consulting of the General Electric Company, Schenectady, New York, under the direction of Mr. Allan Rubenstein. The General Electric Project Manager was Mr. Myron Hurlbut of Industrial Automation Systems Department, Albany, New York.

Certain work aimed at improving Test Bed Technology has been performed by other contracts with Project 6201 performing integrating functions. This work consisted of enhancements to Test Bed software and establishment and operation of Test Bed hardware and communications for developers and other users. Documentation relating to the Test Bed from all of these contractors and projects have been integrated under Project 6201 for publication and treatment as an integrated set of documents. The particular contributors to each document are noted on the Report Documentation Page (DD1473). A listing and description of the entire project documentation system and how they are related is contained in document FTR620100001, Project Overview.

The subcontractors and their contributing activities were as follows:

TASK 4.2

Subcontractors	Role
Boeing Military Aircraft Company (BMAC)	Reviewer
D. Appleton Company (DACOM)	Responsible for IDEF support, state-of-the-art literature search
General Dynamics/ Ft. Worth	Responsible for factory view function and information models

Subcontractors	Role
Illinois Institute of Technology	Responsible for factory view function research (IITRI) and information models of small and medium-size business
North American Rockwell	Reviewer
Northrop Corporation	Responsible for factory view function and information models
Pritsker and Associates	Responsible for IDEF2 support
SofTech	Responsible for IDEF0 support

TASKS 4.3 - 4.9 (TEST BED)

Subcontractors	Role
Boeing Military Aircraft Company (BMAC)	Responsible for consultation on applications of the technology and on IBM computer technology.
Computer Technology Associates (CTA)	Assisted in the areas of communications systems, system design and integration methodology, and design of the Network Transaction Manager.
Control Data Corporation (CDC)	Responsible for the Common Data Model (CDM) implementation and part of the CDM design (shared with DACOM).
D. Appleton Company (DACOM)	Responsible for the overall CDM Subsystem design integration and test plan, as well as part of the design of the CDM (shared with CDC). DACOM also developed the Integration Methodology and did the schema mappings for the Application Subsystems.

Subcontractors	Role
Digital Equipment Corporation (DEC)	Consulting and support of the performance testing and on DEC software and computer systems operation.
McDonnell Douglas Automation Company (McAuto)	Responsible for the support and enhancements to the Network Transaction Manager Subsystem during 1984/1985 period.
On-Line Software International (OSI)	Responsible for programming the Communications Subsystem on the IBM and for consulting on the IBM.
Rath and Strong Systems Products (RSSP) (In 1985 became McCormack & Dodge)	Responsible for assistance in the implementation and use of the MRP II package (PIOS) that they supplied.
SofTech, Inc.	Responsible for the design and implementation of the Network Transaction Manager (NTM) in 1981/1984 period.
Software Performance Engineering (SPE)	Responsible for directing the work on performance evaluation and analysis.
Structural Dynamics Research Corporation (SDRC)	Responsible for the User Interface and Virtual Terminal Interface Subsystems.

Subcontractors and other prime contractors under other projects who have contributed to Test Bed Technology, their contributing activities and responsible projects are as follows:

Subcontractors	Role
General Dynamics/ Ft. Worth	Responsible for factory view

v

Contractors	ICAM Project	Contributing Activities
Boeing Military Aircraft Company (BMAC)	1701, 2201, 2202	Enhancements for IBM node use. Technology Transfer to Integrated Sheet Metal Center (ISMC)
Control Data Corporation (CDC)	1502, 1701	IISS enhancements to Common Data Model Processor (CDMP)
D. Appleton Company (DACOM)	1502	IISS enhancements to Integration Methodology
General Electric	1502	Operation of the Test Bed and communications equipment.
Hughes Aircraft Company (HAC)	1701	Test Bed enhancements
Structural Dynamics Research Corporation (SDRC)	1502, 1701, 1703	IISS enhancements to User Interface/Virtual Terminal Interface (UI/VTI)
Systran	1502	Test Bed enhancements. Operation of Test Bed.

TABLE OF CONTENTS

TABLE OF CONTENTS (Continued)

TABLE OF CONTENTS (Continued)

LIST OF ILLUSTRATIONS

SECTION 1

SCOPE

1.1 Identification

This specification establishes the performance, development, test, and qualification requirements of the computer program identified as the Communications Subsystem, hereinafter referred to as COMM. COMM is one configuration item of the Integrated Information Support System (IISS).

1.2 Functional Summary

The COMM Computer Program Configuration Item (CPCI) provides a mechanism for transferring messages (data and control) between two tasks. These tasks can be executing on the same computer or on different computers. For the latter, the two tasks are Network Transaction Managers (NTM's).

The major functions of the COMM are:

1. Interhost Communications: Interhost Communications is responsible for moving variable length messages without error between computers. This function receives messages from an NTM, passes the messages to a local area network (LAN), receives messages from the LAN, and passes the messages to an NTM.

2. Interprocess Communications: Interprocess Communications allows the NTM to receive messages from and send messages to programs on the same computer. The programs may be another portion on the NTM, COMM, UI/VTI, the precompiler, NDDL, or user application programs. This function also allows programs to use timers and to process fatal errors.

SECTION 2

DOCUMENTS

2.1 Applicable Documents

The following documents were used in the definition of the COMM specification.

2.1.1 Specifications

[1] Control Data Corporation and D. Appleton Co., Inc.; IISS Test Bed CDM Needs Analysis, 7 June 1982; IISS Test Bed CDM Environment, 7 June 1982; IISS Test Bed CDM System Requirements, 7 June 1982.

[2] General Electric Co., Test Bed System Requirement Document (Draft), Revised 23 August 1982.

[3] ICAM Computer-Based Information System (CBIS) System Requirements Document (Draft), 10 September 1981, CI #SRD3101400000.

[4] General Electric Co., Test Bed System Specification (Draft), 23 August 1982.

[5] General Electric Co., Test Bed System Design Specification, 7 February 1983.

2.1.2 Standards

[6] American National Standards Committee X3, American National Dictionary for Information Processing, X3/TR-1-77, September 1977.

[7] ICAM Documentation Standards, 28 December 1981, IDS150120000A.

[8] SofTech, Inc., ICAM Test Bed Interim Standards and Procedures, 31 May 1982; ISP620150000.

[9] General Electric Co., IISS Software Development Guidelines/Conventions (Draft), 23 August 1982.

2.1.3 Other

[10] ICAM Program Office, The Integrated Sheet Metal
     Center, 30 September 1981.

[11] ICAM Program Office, The Role of the ICAM Test Bed
     and Integrated Information Support System (Draft), 18
     May 1982.

[12] SofTech, Inc., IISS Response to CBIS Requirements and
     'Threads': SofTech Reactions, 18 March 1982.

[13] Digital, VAX-11 Architecture Handbook, Digital
     Equipment Corp., Maynard, MA, 1979.

[14] IBM, A Guide to the IBM 3031 Processor Complex and
     Attached Processor Complex of System/370, GC20-18
     54-3; System/370 Principles of Operation, GA22-70000.

[15] Honeywell, Level 6 Minicomputer Systems Handbook.

[16] Digital, VAX/VMS System Services Reference Manual,
     AA-D018B-TE; VAX-11 Information Directory and Index,
     AA-D016D-TE.

[17] Users' Manual, IDBMS (2.0) Users' Manual, June 1980.

[18] Systems Users' Manual, IDBMS (2.0) System Users'
     Manual, June 1980; IBM, OS/VS2 MVS Supervisor
     Services and Macro Instructions; GC28-0683-2.

[19] Honeywell, Level 6 GCOS MOD600 System Concepts,
     CB50-02.

SECTION 3

REQUIREMENTS

This section includes functional and performance requirements for the COMM. In addition, it defines the COMM interfaces to other IISS CPCI's.

3.1 Communications Subsystem

The Communications Subsystem is primarily used to transfer messages (as opposed to files) between Network Transaction Managers on two different host computers. (The transfer of messages between tasks on the same computer is accomplished through Interprocess Communication Primitives.) Two copies of the COMM (the name of a single occurrence of the Communication Subsystem program) reside on each host, and each copy communicates between the NTM and a copy of COMM residing on one of the other hosts (see Figure 3-1). COMM communicates with the NTM via the IISS Interprocess Communication Primitives.

The requirement to implement IISS as a system residing on three computers has caused many system dependent issues to surface. The system design goal is to isolate system dependent coding by creating primitives which hide the system dependent programming from the IISS subsystems. The primitives used by COMM are:

1. Interprocess Communication (IPC)
2. Interhost Communication (IHC)

3.1.1 Communication Subsystem Constraints

The IISS hardware consists of a Local Area Network (LAN) and its interface into each computer (see Figure 3-2). The terminal interface standard is the interface into the LAN. The interface to the IBM is handled through an IBM 3271 station emulator (cluster controller) that manages the asynchronous to synchronous, the EBCDIC to ASCII character set, and the RS232C to 3270 protocol conversions.

The IISS protocol has permanent virtual circuits established between each pair of computers by the LAN when it is started (see Figure 3-1). By placing this requirement on the LAN, the overhead of having the communication software establish

Figure 3-1. IISS Test Bed Communications Subsystem (Structural Schematic)
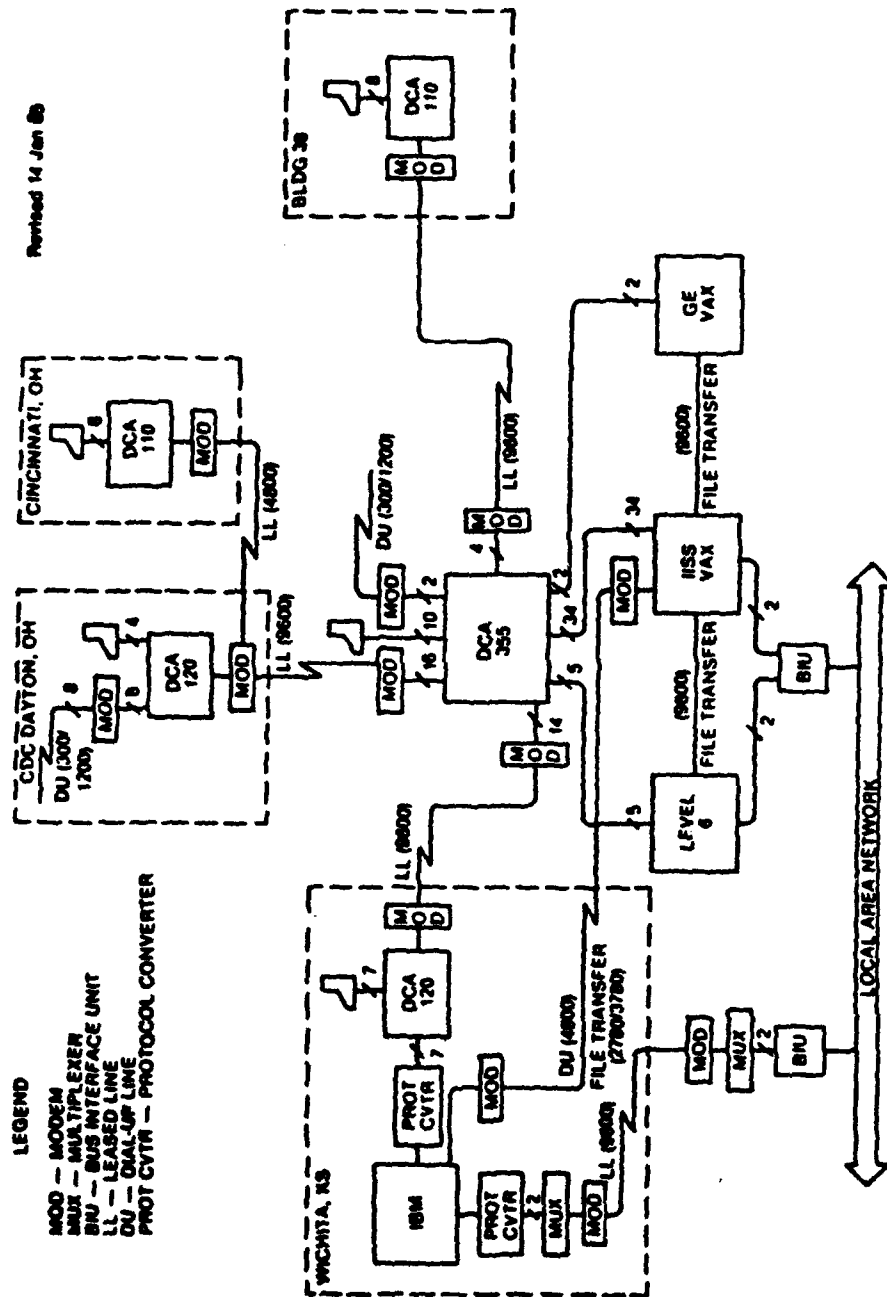
Figure 3-2. IISS Hardware Configuration

the circuit for each message of each session is removed. The COMM will be attached to each line when the computer is started or when IISS is initiated.

The communication subsystem is constrained in the following ways:

1. It must be implemented on each of the following systems with no modifications or additions to the operating systems or communication drivers.

VAX	–	VMS
Honeywell Level 6	–	MOD400
IBM 3084	–	MVS

2. A Local Area Network (LAN) will be used to interface these three systems. Terminal interfaces must be used since these three systems support no other way to uniformly interface to a LAN. The terminal interface to the IBM computer can be supported by using an IBM 3271 station emulator.

3. An MRP program has been chosen to run on IBM using the CICS sub-operating system. CICS supports only COBOL, PL-1, and assembly language. Of these three choices, COBOL has been selected as the implementation language. (This restriction was lifted in 1984.)

4. Programming in assembly language is to be avoided to the greatest extent possible.

These constraints place the following constraints on a data transmission protocol:

1. Full-duplex transmission is not supported.
2. Binary data transmission is supported with data translation.
3. Variable length data messages must be terminated by a carriage return.
4. Eight bit characters are not supported (seven bit characters with the eighth bit used for parity is supported).
5. Message headers, binary data translation, and the longitudinal redundancy checking technique have been chosen to allow for a COBOL implementation.

The communication protocol described in the detailed
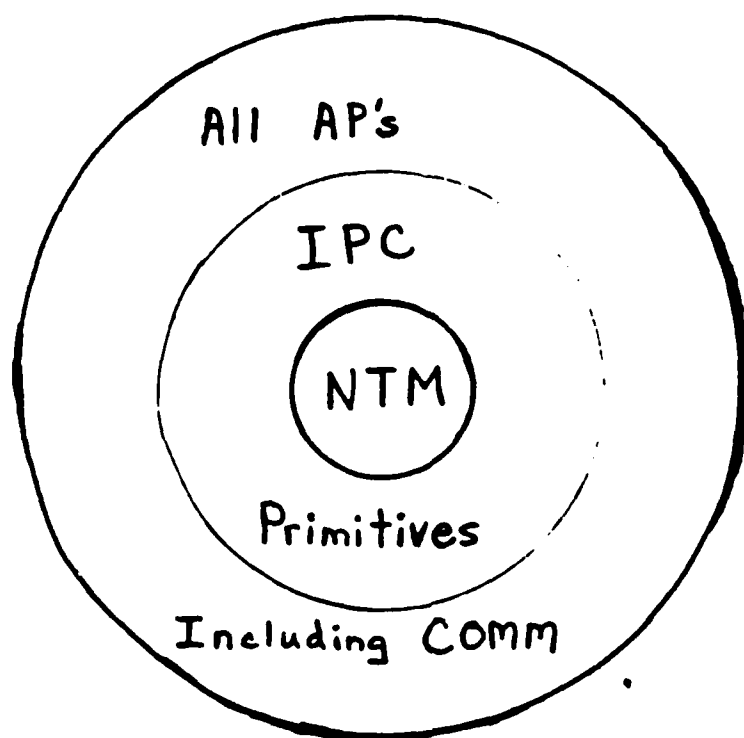
Figure 3-3.   Interprocess Communication Architecture

functional requirements section satisfies these constraints and provides for point-to-point communication between any two of the three NTM's using three permanent virtual circuits on the Local Area Network.

The Interprocess Communication Primitives (IPC) are the mechanisms that the NTM will use to transfer data between itself and the application programs (AP) in the computer (see Figure 3-3). This approach removes the necessity of re-writing the NTM for each computer in the IISS configuration since the system-dependent software will be in the primitives.

Because of the highly system-dependent nature of the communication primitives, they must be implemented three times - once on each computer.

3.1.2 Interface Requirements

The Communication Subsystem consists of two copies of COMM, one resident in each host, enabling two NTM's to communicate with each other. In this sense the communication subsystem interfaces only with the NTM. This interface is accomplished with IPC's. To initiate communication with the NTM, COMM must use an NTM runtime service called INICOM. This service must supply to COMM its input communication mailbox name. COMM must call TRMNAT upon termination in order to stop all further communication with the NTM.

Two types of messages are received by COMM from the NTM:

1. Data messages (both binary and native character set)
2. Control messages

A field in the NTM message header indicates whether or not a message is a data message or a control message. If it is a data message, another field in the header determines whether the data message is "native" or "binary." If it is a control message, the type is included in the message header.

Two types of messages are sent to the NTM by COMM:

1. Data messages (both binary and native character set)
2. Status messages (includes statistics)

The communications subsystem interfaces with the LAN via the standard RS232C terminal interface.

3.1.2.1 Interface Block Diagram

The structural schematic for the communication subsystem is depicted in Figure 3-1. This shows the COMM interfaces to the NTM and the LAN.

3.1.2.2 Detailed Interface Definition of COMM

The Communication Subsystem receives messages from and sends messages to the NTM via the IPC's. Each message contains message data and a message header as described by the NTM. The COMM uses the following fields from the header:

- Destination AP Name
- Message Type
- Binary/Native Flag
- Priority Flag

Upon receiving messages from the NTM, the COMM checks for a control message by checking the destination AP name for COMM. If it is COMM, processing is determined by a message type of startup link (SL) or shutdown link (SD) or terminates (TR). If the destination is not COMM, COMM sends the message according to the binary/native flag.

Messages sent to the NTM from COMM are control messages or data messages. Control messages have the destination set equal to NTM, the priority flag equal to high, and are put in the high priority input queue for NTM. The message type may be link active (LA), link failure (LF) or recoverable error (RE). The data portion of the recoverable error message contains the error number. Data messages have the destination set equal to whatever it was on input to COMM. These messages are put in the correct NTM input queue according to the priority flag.

3.1.2.3 Detailed Interface Definition of Primitives

The VAX implementation of the primitives uses a combination of COBOL and FORTRAN. All system services are called using FORTRAN with most of the other codes such as error checking being done in COBOL.

The Level 6 implementation of the primitives uses a combination of COBOL and Assembly Language. All system services are called using Assembly Language because the Level 6 only supports an Assembly Language interface to system services.

In the IBM system, IISS is implemented in the Assembler and interfaces to the MVS operating system.

3.2 Detailed Functional Requirements

The node tree shown in Figure 3-4 illustrates the COMM functions currently defined.

COMMUNICATION
SUBSYSTEM

INTERHOST
COMMUNICATION

INTERPROCESS
COMMUNICATION
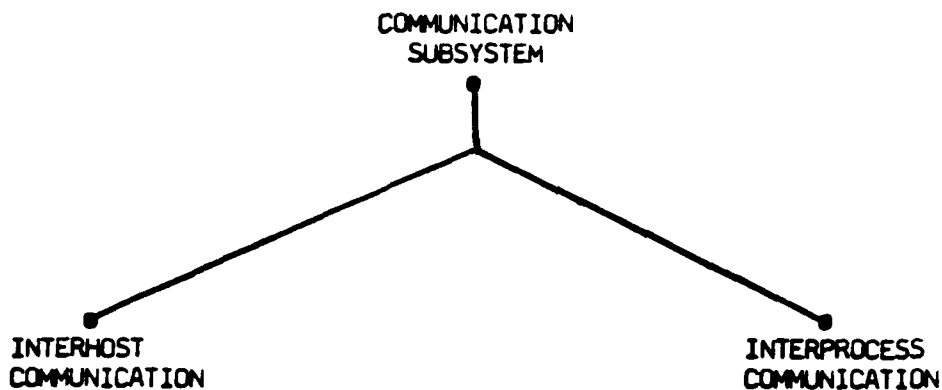
Figure 3-4.   COMM Configuration Tree

Descriptions of these functions may be found in the following paragraphs.

3.2.1   Communication Protocol - Characteristics and Description

A simple scenario for sending a message between NTM's is depicted in Figure 3-5.

3.2.1.1   Characteristics of the Communication Protocol

1.   Uses asynchronous communication lines.

2.   Contention system with one end point designated as primary and the other as secondary.

3.   Point-to-point.

4.   Half-duplex (uses full-duplex communication lines).

5.   Interleaved data transmission.

6. Uses the ASCII character set (excluding control characters 000-037 octal).

7. Error detection and correction by retransmission.

8. Byte stuffing is used to send the control characters. An exclamation mark (!) precedes a translated control character. The control character can then be reconstructed by the receiving COMM program.

9. Eighth bit used for even parity.

10. Accepts variable length NTM messages.

11. All messages terminated by carriage return.

12. Large messages are segmented into packets and reassembled by the receiving COMM.

13. Transmits variable length communication blocks.

14. Symmetric protocol with contention. The retry timing in case of simultaneous line bids is 1 second for the primary endpoint and 5 seconds for the secondary endpoint.

15. Retries a configurable number of times before reporting a link failure to the NTM. Currently, the number is three.

16. Master/Slave relationship determined by the endpoint that successfully bids for the line. Successful bidder becomes the master.

17. Data may be transmitted by either the master or the slave endpoint.

18. All timing is performed by the master. Time out interval is 3 seconds.

19. End of transmission bit may only be sent by the master and only when neither master nor slave has a
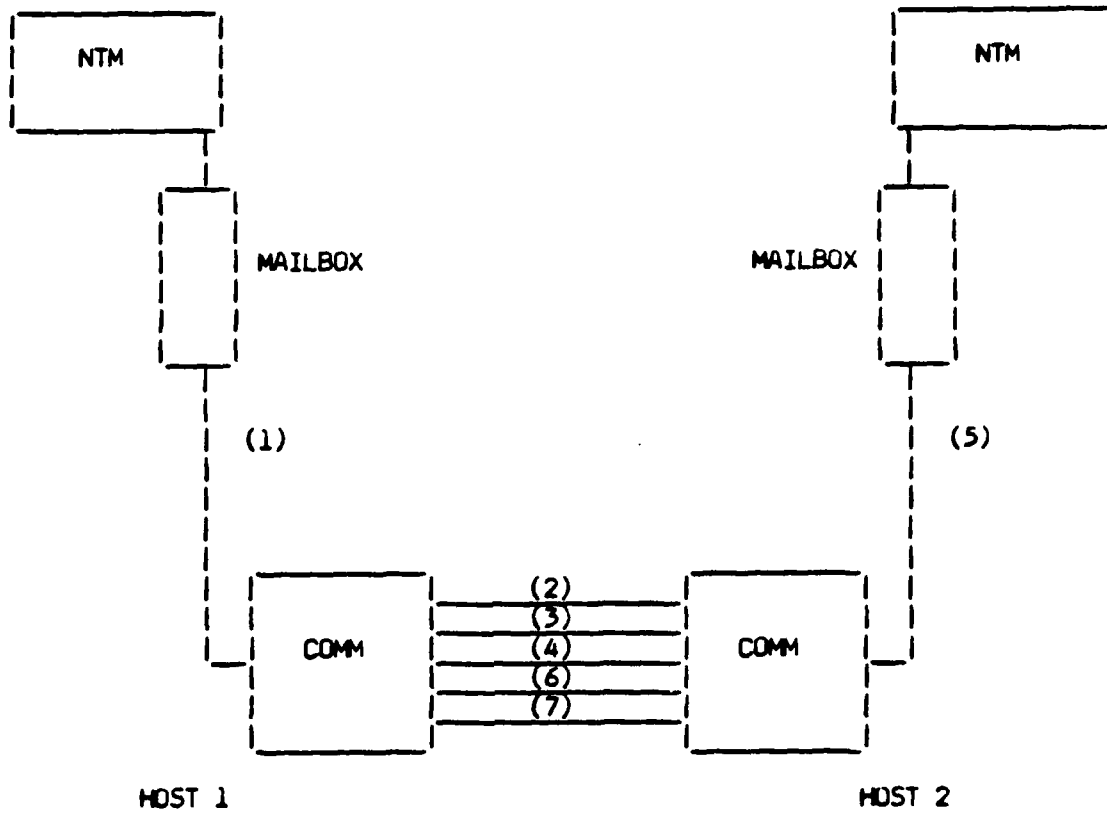
(1) Message received
(2) Line bid
(3) Grant line
(4) Message sent
(5) Message delivered
(6) ACK
(7) EOT

Figure 3-5. Simple Scenario for Sending a Message Between NTM's

3-10

requirement to send data.

3.2.1.2 Sequence Number Handling

The send sequence number and the receive sequence number
are used to prevent either duplication or loss of communiction
blocks. The first data block following a successful line bid is
sent with a send sequence number of 1. Subsequent blocks are
sent with the send sequence number cycling from 2 to 3 and back
to 1. The received sequence number is used in reply to a
sending COMM. A receiving COMM returns the send sequence number
as the received sequence number after forwarding the data to the
local NTM.

3.2.1.2.1 Send Sequence Number Checking

If the send sequence number is as expected, then the data
is stored and that number is returned as the received sequence
number. The expected send sequence number is incremented by
one.

If the send sequence number is one less than expected, then
that number is returned as the received sequence number.
However, the data is not stored since it was stored on receipt
of the prior message (which must have been repeated). The send
sequence number should never be one more than expected.

3.2.1.2.2 Receive Sequence Number Checking

For a given copy of COMM the received sequence number from
the correspondent COMM should be the same as its prior send
sequence number indicating that the correspondent COMM received
the communication block correctly. COMM then increments the
send sequence number for the next block.

If the received sequence number is one less than the prior
send sequence number, then COMM retransmits the data using the
same send sequence number.

The received sequence number should never be one more than
the prior send sequence number. If it is, COMM sends a negative
acknowledgment indicating the error.

3.2.1.3 Protocol for Binary Data Transmission

When an NTM is required to send binary data using this
transmission protocol, the binary information will be expanded

into an acceptable set of characters (0-9 and A-F), transmitted as characters, and transformed to binary upon receipt from the LAN. The following data translation algorithm will be applied to the data by the communication subsystem.

Each byte of data is transmitted as two bytes of ASCII data. The first byte represents the higher order 4 bits of data; the second, the last 4 bits.

This translation is reversed by the receiving station in order to rebuild the original NTM message.

3.2.1.4   Interface Between IBM and non-IBM Computers

When using standard terminal I/O drivers to transmit data between the IBM and non-IBM computers, there are four problems that must be considered:

1. Removing control characters from the message before sending it and inserting them back into the message after it has been received.

2. Using one character set to compute the longitudinal redundancy check on all computers.

3. Conversion or translation of certain characters because of unique problems. For example, spaces are converted to cursor control by protocol converters so spaces must be replaced by a special code before transmission.

4. Checking for characters in EBCDIC that do not have an equivalent character in ASCII.

These problems are solved in the three subroutines, KMINDA, EXOUDA, and KLCLRC, in the Communication Subsystem through the use of two tables. The tables are found in the include files CTLASC and ASCII on non-IBM computers and CTLEBC and EBCDIC on the IBM.

3.2.1.4.1   Non-IBM Computers

The CTLASC and ASCII include files are used in computers whose native character specification is ASCII. The table in the ASCII include file is used by the EXOUDA and KLCLRC subroutines to determine if there is an equivalent EBCDIC character, and, if it is a control character, what the substitution code is.

3-12

The table contains positive, negative, and zero values. The negative values indicate that the character being processed is a control character or a character that requires special conversion for transmission. The code to be used in the message is the negated negative value found in the table. If the value is positive, the original character is used in the message. If the value is zero, there is no equivalent character in the EBCDIC character set.

The table in the CTLASC include file is used by the KMINDA subroutine to restore the original message. Through the table, KMINDA determines the correct control character or special character to be inserted into the message in place of the code character found in the message. The code consists of alphanumeric characters that will not cause terminal I/O drivers to react in a special manner. The flag in the message that indicates a code character follows is the (!).

3.2.1.4.2 IBM Computers

The CTLEBC and EBCDIC include files are used in the IBM computers where the native character specification is EBCDIC. The table in the EBCDIC include file is used for two functions. In EXOUDA it is used to determine if there is a comparable ASCII character, and, if there is an ASCII equivalent, is it a control character or a special character. In KLCLRC the table in EBCDIC is used to convert all the characters in the message to ASCII in order to perform the sum for the longitudinal redundancy check. (Negative values in the table are never used in KLCLRC because conversion is done by EXOUDA before KLCLRC is called.)

The contents of the table in the EBCDIC include file on the IBM have the same definitions for positive, negative, and zero values as its counterpart file, ASCII, does in the non-IBM environment.

The table in the CTLEBC include file is used by the KMINDA subroutine in the Communication Subsystem on the IBM in exactly the same manner as KMINDA in the Communication Subsystem on the non-IBM computers.

3.2.2 Message Format

Each message contains seven characters in addition to the data characters. The first three characters are called the header, and the last four characters are called the trailer.

This is depicted in Figure 3-6.

Header Character #1   —   Send Data Sequence Number Byte

   ASCII Character
         0   No send data message
         1   Send Sequence Number 1
         2   Send Sequence Number 2
         3   Send Sequence Number 3

                     Source sequence number cycles from
                     1 to 2 to 3 and back to 1 when
                     data is being transmitted.

Header Character #2   —   Received Data Sequence Number Byte

   ASCII Character
         0   No received data message
         1   Received Sequence Number 1
         2   Received Sequence Number 2
         3   Received Sequence Number 3

Header Character #3   —   Control Byte

   ASCII Character

The following 8 control chars do not accompany a data msg.
         0   Positive Acknowledgment
         1   Line Bid
         2   On-Line
         3   End of Transmission
         4   NAK - Parity Error
         5   NAK - No Buffer Space
         6   NAK - Bad Receive Sequence Number
         7   NAK - Bad Send Sequence Number

The following 4 control chars always accompany a data msg.
         A   ACK - Sending Native Data
         B   ACK - Sending Binary Data
         C   ACK - Sending Continued Native Msg.
         D   ACK - Sending Continued Binary Msg.

Trailer Characters
  #1, 2, & 3   —   Block Check

These three characters contain a message block check. The block
check is computed by adding all bytes in the message (including

the header bytes). This block check could be up to 18 bits in size for a message block of size 2048 bytes. The 18 bit additive result is split into three 6 bit quantities. Each 6 bit quantity is represented by an ASCII character, with the three characters being stored as the last characters in the message (high order byte occurring first).

The block check characters occur only in messages containing data.

Trailer Character #4   –   Carriage Return

```
-----------------------------------------------------------------
| Send  |  Rcv.  | Ctrl. |       |      |      |      |  | C |
| Seq.  |  Seq.  | Byte  | DATA  | Bcc  | Bcc  | Bcc  |  | R |
| No.   |  No.   |       |       |  1   |  2   |  3   |  |   |
-----------------------------------------------------------------
```

or

```
---------------------------------------------
| Send  |  Rcv.  | Ctrl. |  C  |
| Seq.  |  Seq.  | Byte  |  R  |
| No.   |  No.   |       |     |
---------------------------------------------
```

SEND SEQUENCE   RECEIVE SEQUENCE NUMBER

    0 - No send data   0 - No received data
    1 - Send sequence   1 - Receive sequence
          number 1   number 1
    2 - Send sequence   2 - Receive sequence
          number 2   number 2
    3 - Send sequence   3 - Receive sequence
          number 3   number 3

CONTROL BYTE

        THE FOLLOWING CHARACTERS DO NOT ACCOMPANY DATA

        0 - Positive acknowledgment
        1 - Line bid
        2 - End of transmission
        3 - NAK - Block check error
        4 - NAK - NTM input mailbox full
        5 - NAK - Bad send sequence number
        6 - NAK - Bad receive sequence number

        THE FOLLOWING CHARACTERS ALWAYS ACCOMPANY DATA

        A - ACK - Sending native data
        B - ACK - Sending binary data
        C - ACK - Sending continued native data
        D - ACK - Sending continued binary data

             Figure 3-6. Communication Block Diagram

3.2.3 Timing

3.2.3.1 Line Idle

When the line is idle, the line bid message may be sent by either endpoint. If a collision of line bids occur or if there is no response, the primary waits for 1 second before retransmitting a line bid, and the secondary waits for 5 seconds.

3.2.3.2 Master

The endpoint successfully bidding for the line becomes the master. The master waits for 5 seconds for a response from a slave. A time-out causes retransmission of the prior communication block.

3.2.3.3 Slave

The slave waits for

(Maximum number of retries x 5 secs) + 5 seconds

for a response from the master. If data is not received within this time, the slave assumes an end of transmission was missed and returns to an idle state. If a partial message is being assembled, a link failure is reported and the partial message is discarded.

3.2.3.4 NTM Input Mailbox Full

When a copy of COMM discovers that the NTM input mailbox is full, it waits for one second and tries again to place the message in the mailbox. If the condition still exists, a NAK message is returned to the correspondent COMM. The correspondent COMM sends an error status message to its local NTM and retransmits the communication block.

3.2.4 Block Check Computation

The block check is computed by adding all bytes in the communication block excluding the block check itself and the carriage return. This could account for an 18 bit block check for a message block of size 2048 bytes. The 18 bit additive result is split into three 6 bit quantities. Each 6 bit quantity is represented by a valid ASCII character with the

three characters being stored as the last three characters in the message (high order byte occurring first).

3.2.5 Control Characters in Data

The 32 ASCII control characters (octal 0-37) affect the behavior of the I/O handlers and therefore cannot be allowed in the data stream. Byte stuffing will be used to convert each control character to a valid ASCII character proceeded by a usable special character such as data link escape.

3.2.6 Communication Subsystem Primitives

The design of the COMM is made up of a large generic portion that will be the same for all computers, and a small host-specific part (primitives) that is specially developed for each computer. The host-specific part is a group of routines called the Interhost Communication Primitives (IHC's). The following is a description of these routines.

3.2.6.1 Initialize Communication Port Interface

Calling Sequence:

```
CALL INILAN USING PORT-NAME,
               RCV-BLOCK,
               XMIT-BLOCK,
               EVENT-BLOCK-nn,
               STATUS.
```

Description:

INILAN moves the PORT-NAME or some system dependent equivalent to the appropriate storage in the XMIT and RCV blocks. If initialization for some system services associated with the port is required, it is performed in this primitive. INILAN also initializes the XMIT and RCV blocks with character zeros.

Inputs:

```
PORT-NAME
RCV-BLOCK
XMIT-BLOCK
EVENT-BLOCK-nn
```

Outputs:

STATUS

Possible Status Conditions:

Successful
System dependent errors

3.2.6.2 Send a Message to a Terminal Port

Calling Sequence:

```
CALL "XMTLAN" USING XMIT-BLOCK,
             EVENT-BLOCK-nn,
             FLAGS,
             STATUS.
```

Description:

XMTLAN outputs a message to a given terminal port.

Inputs:

XMIT-BLOCK
EVENT-BLOCK-nn
FLAGS

Outputs:

STATUS

Possible Status Conditions:

Successful
Number of bytes zero
Number of bytes greater than maximum
Receive LAN outstanding
System dependent errors

Notes:

1. The XMIT-BLOCK contains the port name, the number of
   bytes to be transmitted, and the buffer with the
   message. (See the next section for a detailed
   description of the XMIT-BLOCK.)

2. A good status indicates that the message has been accepted for transmission. It does <u>not</u> mean a successful transmission.

3. The number of bytes to be transmitted must be at least one and no more than 1024. The maximum number of bytes is a communication variable that can be changed.

4. The event block that is passed to XMTLAN must be the same as the one that is passed to RCVLAN and GETLAN.

3.2.6.3 Receive a Message from a Terminal Port

Calling Sequence:

CALL "RCVLAN" USING RCV-BLOCK,
              EVENT-NUMBER,
              EVENT-BLOCK-nn,
              FLAGS,
              STATUS.

Description:

RCVLAN informs the operating system that the program will accept a message from the given terminal port.

Inputs:

RCV-BLOCK
EVENT-NUMBER
EVENT-BLOCK-nn

Outputs:

STATUS

Possible Status Conditions:

Successful
Only one receive outstanding permitted
Event number zero
Event number greater than maximum
System dependent errors

Notes:

1. Only one receive may be outstanding for a given
    terminal port.

2. The event number may have a value of 01 through 22. A
    value of zero or 23 through 99 causes an error status
    to be returned.   The maximum number of events
    outstanding that can be waited on at any one time is
    22 because COBOL on the Level 6 limits the number of
    arguments in a calling sequence to 25.

3. The event number must be unique.

4. The value of the event number is the priority of the
    receive message request in relation to the other
    outstanding requests. The lower the value, the higher
    the priority.

5. The RCV-BLOCK contains the port name, the buffer size,
    and the buffer into which the message will be stored.
    (See the next section for a detailed description of
    the RCV-BLOCK.)

6. The RCV-BLOCK that is passed to RCVLAN must be the
    same one that is passed to GETLAN when the program
    actually gets the message from the given terminal
    port.

7. The event block that is passed to RCVLAN must be the
    same one that is passed to GETLAN when the program
    actually gets the message from the given terminal
    port.   The event block is also the same one that is
    passed to XMTLAN to send a message to the port.

3.2.6.4. Get a Message from a Terminal Port

Calling Sequence:

CALL "GETLAN" USING RCV-BLOCK,
                EVENT-BLOCK-nn,
                STATUS.

Description:

GETLAN accepts the message that was received from the given
terminal port and moves it into the given buffer.

Inputs:

>RVC-BLOCK
>EVENT-BLOCK-nn

Outputs:

>STATUS

Possible Status Conditions:

>Successful
>Receive not satisfied
>Not a LAN event block
>No receive outstanding
>Buffer size zero
>Buffer size greater than maximum
>Buffer too small
>System dependent errors

Notes:

1. The RCV-BLOCK contains the port name, the buffer size,
    the buffer, and a location into which the number of
    bytes in the message is stored by GETLAN.

2. The RVC-BLOCK that is passed to GETLAN must be the
    same one that was passed to RCVLAN for the given
    terminal port.

3. The event block that is passed to GETLAN must be the
    same one that was passed to RCVLAN for the given
    terminal port. The event block is also the same one
    that is passed to XMTLAN to send a message to the
    port.

4. If the buffer size is too small for the entire
    message, an error status is returned and the message
    is lost. There is no longer a receive outstanding.

5. If no receive is outstanding for the given terminal
    port, an error status is returned.

3.2.6.5 Cancel a Receive Request from a Terminal Port

Calling Sequence:

```
CALL "CNLLAN" USING RCV-BLOCK,
              EVENT-BLOCK-nn,
              STATUS.
```

Description:

CNLLAN removes the receive outstanding for a given terminal port.

Inputs:

        RCV-BLOCK
        EVENT-BLOCK-nn

Outputs:

        STATUS

Possible Status Conditions:

        Successful
        No receive outstanding
        Not a LAN event block
        System dependent errors

Notes:
1. The RCV-BLOCK that is passed to CNLLAN must be the same one that was passed to RCVLAN when the receive was requested.

2. The event block that is passed to CNLLAN must be the same one that was passed to RCVLAN when the receive was requested.

3. The RCV-BLOCK contains the port name, the buffer size, the buffer into which the message will be stored, and a location for the actual number of bytes in the message. (See the next section for a detailed description of the RCV-BLOCK.)

4. If there was a message present, it is lost.

3.2.6.6   Terminate Communication Port Interface

Calling Sequence:

```
CALL "TRMLAN" USING PORT-NAME,
              RCV-BLOCK,
              XMIT-BLOCK,
              EVENT-BLOCK-nn,
              STATUS.
```

Description:

TRMLAN is needed for the IBM environment only. It issues VTAM calls to disconnect from the port. The implementation on the other computers is a stub.

Inputs:

```
        PORT-NAME
        RCV-BLOCK
        XMIT-BLOCK
        EVENT-BLOCK-nn
```

Outputs:

```
        STATUS
```

3.2.7   Interprocess Communication Primitives

The IPC primitives are used to transfer messages between tasks on the same computer. They have been designed to localize all host dependent code in several small routines. These routines are referred to as the Interprocess Communication Primitives or IPC's.

3.2.7.1   Create a Mailbox

Calling Sequence:

```
CALL "CRTMBX" USING INPUT-MAILBOX-NAME,
              MAILBOX-SIZE,
              EVENT-BLOCK-nn,
              STATUS.
```

Description:

CRTMBX creates a mailbox through which the program will

receive messages from another program running on the same
computer.

Inputs:

> INPUT-MAILBOX-NAME
> MAILBOX-SIZE
> EVENT-BLOCK-nn

Outputs:

> STATUS

Possible Status Conditions:

> Successful
> Invalid mailbox name
> Mailbox already exists
> Mailbox size zero
> Mailbox size greater than maximum
> Event block not initialized
> System dependent errors

Notes:

1. If the input mailbox has been created previously, an
    error status is returned.

2. The event block that is passed in the CRTMBX call is
    the same event block that is passed when the program
    issues RCVMSG and GETMSG for the input mailbox.

3. The mailbox size is only applicable on the VAX and IBM
    implementations of the CRTMBX primitive. The resource
    wait mode on the VAX must be disabled to permit the
    program to regain control immediately upon detection
    of mailbox full.

3.2.7.2 Send a Message to Another Program

Calling sequence:

```
CALL "SNDMSG" USING  TARGET-MAILBOX-NAME,
                     BUFFER,
                     NUMBER-OF-BYTES,
                     EVENT-BLOCK-nn,
                     STATUS.
```

Description:

SNDMSG sends a message to another program running on the same computer through the input mailbox of the other program. The event block is system dependent storage that is required by SNDMSG. It is not associated with an event that can be waited on.

Inputs:

TARGET-MAILBOX-NAME
BUFFER
NUMBER-OF-BYTES
EVENT-BLOCK-nn

Outputs:

STATUS

Possible Status Conditions:

Successful
Mailbox not found
Mailbox full
Number of bytes zero
Number of bytes greater than maximum
System dependent errors

Notes:

1. The receiving program must have previously created its input mailbox. If no mailbox exists with the given name, an error status is returned.

2. A good status indicates that the message has been accepted by the operating system for transfer. It does not mean that the message has been received by the other program.

3. If the status of "mailbox full" is returned, the program must retry at a later time.

4. The number of bytes to be sent must be at least one and no more than 2000.

5. The event block must not be in use for an input

mailbox or a timer. An event block associated with a
mailbox is in use if the mailbox has been created but
not deleted. An event block associated with a timer is
in use if the timer has been started but not cancelled
or runout during a WAITnn.

6. A series of SNDMSG calls being directed to a single
    target mailbox should use the same event block. The
    VAX assigns a logical channel ID to a target mailbox
    at the time of the first SNDMSG call and uses that
    channel ID for subsequent SNDMSG calls. This event
    block cannot be used for other purposes until it is
    released (see RELEVB).

3.2.7.3 Receive a Message from Another Program

Calling sequence:

CALL "RCVMSG" USING INPUT-MAILBOX-NAME,
                    EVENT-NUMBER,
                    EVENT-BLOCK-nn,
                    STATUS.

Description:

RCVMSG informs the operating system that the program will
accept a message sent from another program to the given input
mailbox. The program continues executing. The fact that a
message has been sent by another program is obtained with the
WAITnn or GETMSG primitive.

Inputs:

            INPUT-MAILBOX-NAME
            EVENT-NUMBER
            EVENT-BLOCK-nn

Outputs:

            STATUS

Possible Status Conditions:

            Successful
            Invalid event block for mailbox named
            Not a receive event block
            Only one outstanding receive permitted

Event number zero
Event number greater than maximum
System dependent errors

Notes:

1. Only one receive may be outstanding for a given input
    mailbox.

2. The event block that is passed to RCVMSG must be the
    same event block that was passed to CRTMBX when the
    given input mailbox was created.

3. The event number must be unique. (See the description
    of the event number in Section 3.5.2)

4. The value of the event number is the priority of the
    receive message request in relation to the other
    outstanding requests. The lower the value, the higher
    the priority.

5. The event number may have a value of 01 through 22. A
    value of zero or 23 through 99 causes an error status
    to be returned. Because COBOL on the Level 6 limits
    the number of arguments in a calling sequence to 25,
    the maximum number of outstanding events that can be
    waited on at any one time is 22.

3.2.7.4 Get a Message from Another Program

Calling Sequence:

CALL "GETMSG" USING INPUT-MAILBOX-NAME,
                    BUFFER,
                    BUFFER-SIZE,
                    NUMBER-OF-BYTES,
                    EVENT-BLOCK-nn,
                    STATUS.

Description:

GETMSG accepts the message that was sent from another program
running on the same computer and moves it into the given buffer.

Inputs:

INPUT-MAILBOX-NAME
BUFFER-SIZE
EVENT-BLOCK-nn

Outputs:

BUFFER
NUMBER-OF-BYTES
STATUS

Possible Status Conditions:

Successful
Invalid event block for mailbox name
Not a receive event block
No receive outstanding
Receive not satisfied
Buffer too small
Buffer size zero
Buffer size greater than maximum
System dependent errors

Notes:

1. If no receive is outstanding for the given input
   mailbox, an error status is returned.

2. If a receive is outstanding but no message has been
   delivered by the operating system, a status is
   returned indicating that no message has been received.

3. If the buffer size is too small for the entire
   message, an error status is returned and the message
   is lost. An outstanding receive for that mailbox no
   longer exists.

4. The event block that is passed to GETMSG must be the
   same one that was passed to RCVMSG for the given input
   mailbox.

3.2.7.5 Delete a Mailbox

Calling Sequence:

CALL "DELMBX" USING INPUT-MAILBOX-NAME,
                    EVENT-BLOCK-nn,
                    STATUS.

Description:

DELMBX removes the ability to receive messages from another
program through the given input mailbox.

Inputs:

> INPUT-MAILBOX-NAME
> EVENT-BLOCK-nn

Outputs:

> STATUS

Possible Status Conditions:

> Successful
> Invalid event block for mailbox named
> Not a receive event block
> System dependent errors

Notes:

1. The event block that is passed to DELMBX must be the
    same event block that was passed to CRTMBX when the
    given input mailbox was created.

2. If there are messages remaining in the mailbox, they
    are lost.

3. The given event block is re-initialized.

3.2.7.6 Release an Event Block

Calling Sequence:

Call "RELEVB" USING TARGET-MAILBOX-NAME,
                    EVENT-BLOCK-nn,
                    STATUS.

Description:

RELEVB releases an event block which had been used for
sending messages to a target mailbox. The event block is cleared
and may be used for another purpose.

Inputs:

TARGET-MAILBOX-NAME
EVENT-BLOCK-nn

Outputs:

STATUS

Possible Status Conditions:

Invalid event for mailbox named
System dependent errors (VAX only)

Notes:

1. The VAX implementation of RELEVB deassigns the logical
   channel previously assigned to the target mailbox by
   SNDMSG.

2. All logical channels assigned to target mailboxes by
   SNDMSG primitive calls on the VAX are deassigned when
   a process terminates. Therefore, for many
   applications, a call to RELEVB is not required.

3.2.7.7 Start a Timer

Calling Sequence:

CALL "SETTIM" USING TIME-INTERVAL,
                    EVENT-NUMBER,
                    EVENT-BLOCK-nn,
                    STATUS.

Description:

SETTIM requests the operating system start a timer with the
given time interval. The program continues executing. The fact
that the timer has elapsed is obtained from the WAITnn
primitive. Only one timer may be active at a time.

Inputs:

TIME-INTERVAL
EVENT-NUMBER
EVENT-BLOCK-nn

Outputs:

STATUS

Possible Status Conditions:

                Successful
                Time interval zero
                Time interval greater than maximum
                Event number zero
                Event number greater than maximum
                Event block not initialized
                System dependent errors
                Timer already active

Notes:

1. The time interval is given as HHMMSS.

2. The minimun time interval is 000001. The maximum time
    interval is 24 hours, 59 minutes, 59 seconds.

3. The event number must be unique. (See the description
    of the event number in Section 3.5.2)

4. The value of the event number may be 01 through 22. A
    value of zero or 23 through 99 causes an error status
    to be returned. The maximum number of outstanding
    events that may be waited on at any one time is 22
    because COBOL on the Level 6 limits the number of
    arguments in a calling sequence to 25.

5. The value of the event number in relation to the
    values of other event numbers is an indication of
    priority for the WAITnn primitive. The event number
    with the lowest value has the highest priority.

6. The event block that is passed to SETTIM must be the
    same event block that is passed to CNLTIM to cancel
    the timer.

3.2.7.8 Stop a Timer

Calling Sequence:

CALL "CNLTIM" USING EVENT-BLOCK-nn,
                    STATUS.

Description:

CNLTIM cancels the request made to the operating system by SETTIM to be notified when a given time interval has passed.

Inputs:

EVENT-BLOCK-nn

Outputs:

STATUS

Possible Status Conditions:

Successful
Not a timer event block
System dependent errors

Notes:

1. The event block that is passed to CNLTIM must be the same event block that was passed to SETTIM to start the timer.

2. The given event block is re-initialized.

3.2.7.9 Wait for an Event to Occur

Calling Sequence:

CALL "WAITnn" USING EVENT-NUMBER,
                    STATUS,
                    NUMBER-OF-EVENT-BLOCKS,
                    EVENT-BLOCK-01,
                    EVENT-BLOCK-02,

                    .
                    .
                    .
                    EVENT-BLOCK-xx.

where nn is a number from 01 through 22 that indicates the number of event blocks being passed

Description:

WAITnn waits for one of the outstanding requests that are
associated with the list of event blocks to be satisfied. The
event number associated with the completed request is returned
in the EVENT-NUMBER variable.

Inputs:

NUMBER-OF-EVENT-BLOCKS
EVENT-BLOCK-01
EVENT-BLOCK-02
.
.
.
EVENT-BLOCK-xx

Outputs:

EVENT-NUMBER
STATUS

Possible Status Conditions:

Successful
Number of event blocks zero
Number of event blocks greater than
maximum
List of event blocks greater than
number of event blocks
Event numbers not unique
No requests outstanding
System dependent errors

Notes:

1. The order in which the testing for the completion of a
    request is performed is based upon the event number
    associated with each event block. The request with
    the lowest event number has the highest priority;
    therefore, the check for its completion is done first.
    The request with the next lowest event number is
    checked second, and so forth. This process continues
    until a request associated with one of the given event
    blocks has been satisfied.

2. If two requests have been completed, the one with the
    lowest event number is indicated to the calling
    program. The information that another request has been

satisfied is retained by the primitive until the
WAITnn is called again.

3. The maximum number of event blocks that may be passed
    is 22. Because COBOL on the Level 6 limits the number
    of arguments in a calling sequence to 25, the maximum
    number of outstanding events that can be waited on at
    any one time is 22.

4. At least one event block must be passed.

5. Each event block must have a unique event number
    associated with it; otherwise, an error status is
    returned.

6. It is not required to have an outstanding request
    associated with each event block passed to WAITnn.
    Because the primitive is able to differentiate between
    event blocks that have outstanding requests and those
    that do not, the WAITnn primitive may be passed all
    the event blocks that would be needed for the worst
    case. If there are no requests outstanding, however,
    an error status is returned.

7. If an error status is returned, the event number is
    set to character zeros.

8. The 'list of event blocks greater than number of event
    blocks' error is checked only by the IBM.

3.2.7.10 Terminate a Program

Calling sequence:

CALL "ENDRUN".

Description:

ENDRUN calls the appropriate system routine to stop
execution of the particular program without halting the IISS
system.

Inputs:

        None

Outputs:

None
3.2.7.11 Save an Event Indicator

Calling Sequence:
CALL "LOCKEF".

Description:

LOCKEF is applicable only to the VAX. Its purpose is to
obtain event indicator 63 from the system so that none of the
other primitives can use it. LOCKEF is needed to work around a
bug in VAX-11 DBMS. It is called by the Request Processors that
manipulate data store under the VAX-11 DBMS.

Inputs:

None

Outputs:

None

3.2.7.12 Request an Error Be Logged

ERRPRO -- Process Error

Description:

This module is used to process errors. It gets the current
date and time, formats the message and writes the message to the
mailbox ERRMBX. In case of an error, this module will print a
message on the operator hardcopy console. It will not terminate
the processing of the calling program, but will return control
to the calling program regardless of whether the error message
is being written to the mailbox or not. Another task, ERRLOG,
will read the error message from the mailbox and log the message
on a file. See module specification of ERRLOG for details of
this error logging task. The format of an error message is as
shown:

Byte	Data
1 - 2	function code (DA)
3 - 10	current date (YY/MM/DD)

11 - 18	current time (HH:MM:SS)
19 - 24	module name
25 - 29	return status
30 - 89	message description

INTERFACES:

    ENTRY CONDITIONS:

RET-STATUS	PIC X(5).
MODULE-NAME	PIC X(6).
MESG-DESC	PIC X(60).

EXIT CONDITIONS:

    (NONE)

GLOBAL BLOCKS:

    (NONE)

DATA ORGANIZATION:
    LOCAL VARIABLES:

ERR-STATUS   PIC X(5).   --ERROR STATUS

DATABASE INTERACTION:

LIMITATIONS:

    Depending on the operating system, we may have a problem
trying to write a message to the operator console. If so, some
other mechanism will be used to log the error status on a
hardcopy device in case this module encounters an error while
writing to the ERRMBX mailbox.

3.2.7.13 Log an Error

ERRLOG -- PERFORM ERROR LOGGING TO A FILE

Description:

    This program reads a message from mailbox ERRMBX. The
format of the message is as follows:

    Byte   Data

1 - 2	Function code
	e.g., DA -- error message data to be logged to file ERRLOG.DAT CF -- close current version of file and open a new version of the file ST -- close current version of file and terminate processing
3 - 109	Data portion of message -- If the function code is DA, the data portion of the message is formatted as follows:

Byte	Data
3 - 10	current date (YY/MM/DD)
12 - 19	current time (YY/MM/DD)
21 - 35	process name
37 - 42	module name
44 - 48	return status
50 - 109	message description
11, 20, 36, 43, 49	(blank)

Depending on the function code, this routine will branch to the corresponding part of the program. If an error is detected in this module, it will try to display the error on the operator's hard copy device. This module is executed as soon as IISS is brought up. It will check to see if mailbox ERRMBX is already created or not. If not, it will create the mailbox. Otherwise it will set up the event block for a future receive. Mailbox ERRMBX is a temporary mailbox and there is no need to delete or clean up the mailbox when this program is brought up. If there is no message in the mailbox, this program will suspend processing and wait for a message to arrive at mailbox ERRMBX. If the message contains function code ST, this routine will close the current version of the file and terminate processing. If the message contains function code CF, this routine will close the current version of the file and open a new version of the file. Then it continues to read messages from the mailbox. If the message contains function code DA, it will write a record to the ERRLOG.DAT file and continue to read messages from the mailbox.

DATA ORGANIZATION:

   LOCAL VARIABLES:

DATABASE INTERACTION:

ERRLOG.DAT -- this is an indexed file of record size (102) with the following format:

Byte	Data
1 - 8	current date (YY/MM/DD)
9 - 16	current time (YY/MM/DD)
17 - 31	process name
32 - 37	module name
38 - 42	return status
43 - 102	message description

LIMITATIONS:

LEVEL 6 MOD 400 operating system does not keep track of the version number of the file. In order to keep different versions of the file, we may have to rename the current version of the file and then create a new version using ERRPRO.DAT as the file name.

3.3 Special Requirements

3.3.1 Programming Methods

COMM programming methods shall conform to the standards set forth by General Electric in the IISS Software Development Guidelines/Conventions document. Principles of structured design and programming will be adhered to.

3.3.2 Expandability

The design constraints on the communictions subsystem were to follow the ISO reference model, use a local area network, and, if possible, use standard, vendor-supported softwre drivers. The objective is to avoid writing new drivers with all of the maintenance problems that this entails, and to avoid developing any special hardware. Various communications packages and approaches were reviewed and evaluated with respect to their applicability to the local area network usage, the availability for the computers to be used on the Test Bed, and their adherence and extension to the ISO reference model. The final conclusion was that no communication protocol existed or was in the detailed specification stage, even for the lower

levels, to fully satisfy the needs and constraints of the Test
Bed and at the same time not to unduly overburden the system
with capabilities that are not needed for a LAN-based system.

Based on the evaluation of currently available packages,
and in consultation with Computer Technology Associates (active
on the ISO committee), it was decided to develop a protocol
based on the standard "bisynch" protocol for level 2 that would
serve in the interim period and be compatible with the
substitution of new standard protocols when they become
available. Also, local area network vendors will begin
supplying compatible host protocols. In the meantime, the
protocols developed for the project will use standard terminal
drivers supplied by the computer vendors and will interface to
the local area network as a standard terminal, making the
interface completely standard for the LAN.

Since the system dependent software for each host computer
on the IISS is implemented in the primitives, additional types
of computers can be added to the IISS and only the primitives
have to be reimplemented.

3.4 Human Performance

Not Applicable

3.5 Data Description of Arguments Used with the Primitives

3.5.1 Data Description of Arguments for Communication Subsystem Primitives

- Xmit-block

The argument XMIT-BLOCK is a variable name associated with
a block of contiguous storage. It contains the port name, the
number of bytes to be transmitted, and the buffer for the
message. The message consists of the header, the data, and the
longitudinal redundancy check (LRC). The location of the LRC is
the three bytes immediately following the data. XMIT-BLOCK
also contains the buffer size so that both blocks can be
manipulated by the same subroutines. Its description in COBOL
is

```
01 XMIT-BLOCK.
        03 XMIT-PORT-NAME              PIC X(4).
        03 XMIT-NO-OF-CHARS            PIC 9(4).
        03 XMIT-BUFFER-SIZE            PIC 9(4).
```

```
        03 XMIT-BUFFER.
        05 XMIT-HEADER1                    PIC 9.
        05 XMIT-HEADER2                    PIC 9.
        05 XMIT-HEADER3                    PIC X.
        05 XMIT-DATA                PIC X(1021).
        03 XMIT-SEQUENCE-NO                PIC 9
```

- Rcv-block

The argument RCV-BLOCK is a variable name associated with a
contiguous block of storage. It contains the port name, the
number of bytes in the message just received, the maximum number
of bytes the buffer can hold, and the buffer for the message.
Its description in COBOL is

```
    01 RCV-BLOCK.
        03 RCV-PORT-NAME                   PIC X(4).
        03 RCV-NO-OF-CHARS                 PIC 9(4).
        03 RCV-BUFFER-SIZE                 PIC 9(4).
        03 RCV-BUFFER.
            05 RCV-HEADER1                 PIC 9.
            05 RCV-HEADER2                 PIC 9.
            05 RCV-HEADER3                 PIC X.
            05 RCV-DATA                    PIC X(1021).
        03 RCV-SEQUENCE-NO                 PIC 9.
```

- Flags

The argument FLAGS is a variable name associated with a
contiguous block of storage.   It contains data that indicates
whether this version of COMM is primary, what the state of COMM
is, time settings, and the host and target computer indicators.

- Port-name

The argument PORT-NAME contains the alphanumeric characters
required by the operating system to indicate the exact terminal
port the COMM program will use.

```
01 PORT-NAME            PIC X(12)
```

3.5.2   <u>Data Description of Arguments for IPC Primitives</u>

- Input-mailbox-name

The argument INPUT-MAILBOX-NAME contains a 14-character
alphanumeric (no embedded blanks) that is used by the program to
indicate it will accept and process messages from other programs
running on the computer if they are sent to a mailbox with this
14-character label. It is recommended that a different mailbox
name be used to receive messages from different programs. Its
description in COBOL is

       01 input-mailbox-name-x   PIC X(14).

       - Target-mailbox-name

The argument TARGET-MAILBOX-NAME contains the 14-character
alphanumeric that is the input mailbox name for the program to
which the message is to be sent. Its description in COBOL is

       01 target-mailbox-name-x   PIC X(14).

       - Mailbox-size

The argument MAILBOX-SIZE contains the the amount of
storage in bytes the programmer wants allocated by the operating
system for the given mailbox. Its description in COBOL is

       01 mailbox-size-x   PIC 9(5).

       - Buffer

The argument BUFFER is a variable name that is associated
with a given amount of contiguous memory. The amount of memory
should be enough to contain the largest single message that will
be sent or received. Its description in COBOL is

       01 buffer-x   PIC X(2000).

       - Number-of-bytes

The argument NUMBER-OF-BYTES contains the actual number of
characters that, in the case of SNDMSG, are to be sent. In the
case of GETMSG it is the number of bytes that were moved into
the buffer. Its description in COBOL is

       01 number-of-bytes-x   PIC 9(4).

       - Buffer-size

The argument BUFFER-SIZE contains the maximum number of

bytes that can be stored in the buffer by GETMSG. Its description in COBOL is

```
01 buffer-size-x                                    PIC 9(4).
```

- Number-of-event-blocks

The argument NUMBER-OF-EVENT-BLOCKS contains the number of event blocks that are being passed to the WAITnn block. Its description in COBOL is

```
01 number-of-event-blocks-x                         PIC 99.
```

- Time-interval

The argument TIME-INTERVAL contains the number of hours, minutes and seconds the timer is to count before it returns a request complete. Its description in COBOL is

```
01 time-interval-x.
   03 time-in-hours                                 PIC 99.
   03 time-in-minutes                               PIC 99.
   03 time-in-seconds                               PIC 99.
```

The maximum time interval is 24 hours, 59 minutes, 59 seconds.

- Status

The argument STATUS contains a code that indicates whether the primitive was successful or not; and, if it was not, what the problem was. Its description in COBOL is

```
01 status-x                                         PIC X(5)
```

- Event-block

The argument EVENT-BLOCK is a variable name that is associated with a block of contiguous memory to be used by the primitive. It is, therefore, system dependent storage that may not be used by the program. Since it is system dependent, its size will vary from host to host. There are a set of include files that a programmer may use to define the event blocks. The names of the files are

```
EVBK01.system standard suffix
EVBK02. "    "    "
```

EVBK22. " " "

The programmer must decide the maximum number of event
blocks that will be needed by the program at any one time. In
the initialization section of the program, all the event blocks
must be initialized to character zeros. The event block will be
re-initialized by the primitives when the timer runs out or is
cancelled and when a mailbox is deleted.

The event block differs from the other arguments passed to
the primitives. For all the others, the primitives need the
contents or value of the arguments. In the case of the event
block, however, the primitives need its address. Therefore, a
dummy argument for an event block will not work. The following
code illustrates the incorrect passing of event blocks.

```
MOVE EVENT-BLOCK-01 TO EVENT-BLOCK.
PERFORM CREATE-MAILBOX.
        .
        .
        .

MOVE EVENT-BLOCK-02 TO EVENT-BLOCK.
PERFORM CREATE-MAILBOX.
        .
        .
        .

CREATE-MAILBOX.
CALL "CRTMBX" USING INPUT-MAILBOX-NAME,
                    MAILBOX-SIZE,
                    EVENT-BLOCK,
                    STATUS.
        .
        .
        .
```

Programmers should use the CASE statement to invoke the
primitives requiring an event block as an argument.

- Event-number

The argument EVENT-NUMBER is a code that is returned by the
WAITnn primitive to indicate which of the outstanding requests
was satisfied. The code is set by the programmer through the
value in the EVENT-NUMBER passed to the RCVMSG and SETTIM
primitives. The value must be in the range 01 through 22.

For example, a program could create MAILBOX-A, MAILBOX-B and MAILBOX-C and issue a receive on each of the mailboxes with EVENT-NUMBER set to 01, 02 and 03 respectively. If a message was sent by another program to MAILBOX-B, the WAITnn primitive would return 02. Thus, the programmer can use the EVENT-NUMBER as a condition data item to determine what program sent the message.

Its description in COBOL is

01 event-number   PIC 99.

The event number is also used by the WAITnn primitive to determine the order in which the outstanding requests are to be tested. The request with the lowest event number is checked first. The request with the next lowest event number is checked second, and so forth.

3.5.3 Data Description for the Event Block

The contents of the event block are system dependent except for the first two fields. The first field is the event type with a COBOL description of PIC 99. The second field is the event outstanding flag with a COBOL description of PIC 9. Both fields are described in more detail below.

- Event-type

    The possible values for event-types are

    01 IPC Receive
    02 Timer Runout
    03 LAN Receive (Terminal input complete)

- Event-outstanding

    The possible values of event-outstanding codes are

Zeros	No request outstanding
Character One	Request outstanding, wait not complete
Character Two	Request outstanding, wait complete. This state indicates that the request for a receive on either the LAN or a mailbox has been satisfied but that the user has not performed a get. The user is not required to perform a get after a wait or receive.

3-45

3.6 Adaptation Requirements

The COMM must be compiled, linked, and installed on each
host (VAX/VMS, HL 6/GCOS MOD400, IBM/MVS) in the IISS test bed.
These are located at the General Electric facility in Albany,
New York. If a new host type is added to the configuration, the
host specific primitives must be reimplemented for the new host
and relinked with the recompiled nonhost-specific software.

SECTION 4

QUALITY ASSURANCE PROVISIONS

4.1 Introduction and Definition

"Testing" is a systematic process that may be preplanned
and explicitly stated. Test techniques and procedures may be
defined in advance and a sequence of test steps may be
specified. "Debugging" is the process of isolation and
correction of the cause of an error.

"Antibugging" is defined as the philosophy of writing
programs in such a way as to make bugs less likely to occur and
when they do occur, to make them more noticeable to the
programmer and the users. That is, do as much error checking as
is practical and possible in each routine. This approach will
be followed in the COMM.

The quality assurance provisions for test will consist of
the normal testing techniques that are accomplished during the
construction process. They consist of design and code
walk-throughs, unit testing, and integration testing. These
tests will be performed by the design team. Structured design,
design walk-throughs and the incorporation of "antibugging"
facilitate this testing by exposing and addressing problem areas
before they become coded "bugs." A detailed description of the
unit tests for COMM is given in the Unit Test Plan for the
Communication Subsystem, UTP620143000.

The integration testing will entail use of a test NTM
implementation on each host. This test program will send
messages to COMM, read messages from COMM, and print out
results. COMM can also read messages from a terminal and output
messages to a terminal in the absence of a LAN. A simple cable
can also be used to connect the VAX and L6 instead of the LAN.

SECTION 5

PREPARATION FOR DELIVERY

The implementation site for the constructed software will
be the ICAM Integrated Supost System (IISS) Test Bed site
located at General Electric in Albany, NY. The software
associated with each COMM CPCI release will be delivered on a
media which is compatible with the IISS Test Bed. The release
will be clearly identified and will include instructions on
procedures to be followed for installation of the release.

5-1